

GOTO **AARHUS 2021**

#GOTOaar



eficode

Applying Conway

Building organizations that scale

Henrik René Høegh

Consultant



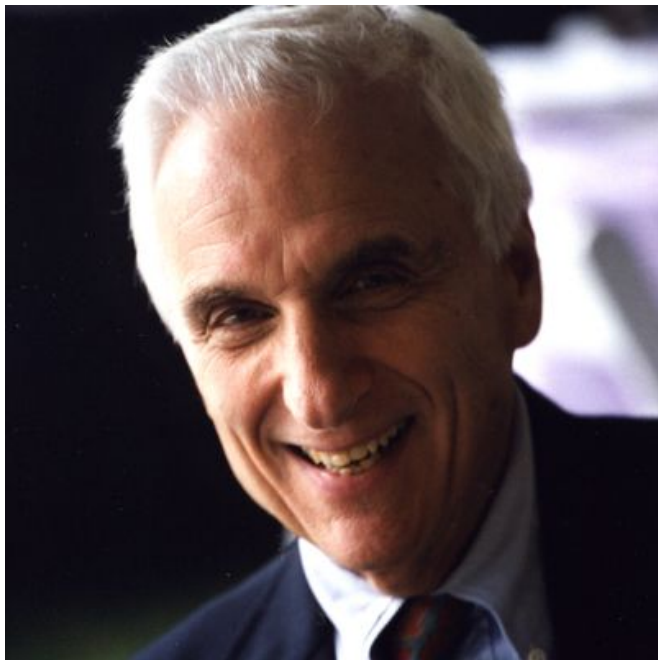
Henrik Høegh

Cloud Native, Scaling organizations

Twitter: @HenrikHoegh

LinkedIn: [linkedin.com/in/henrikrenehoegh](https://www.linkedin.com/in/henrikrenehoegh)

Contact: henrik.hoegh@eficode.com



Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure

Melvin Conway

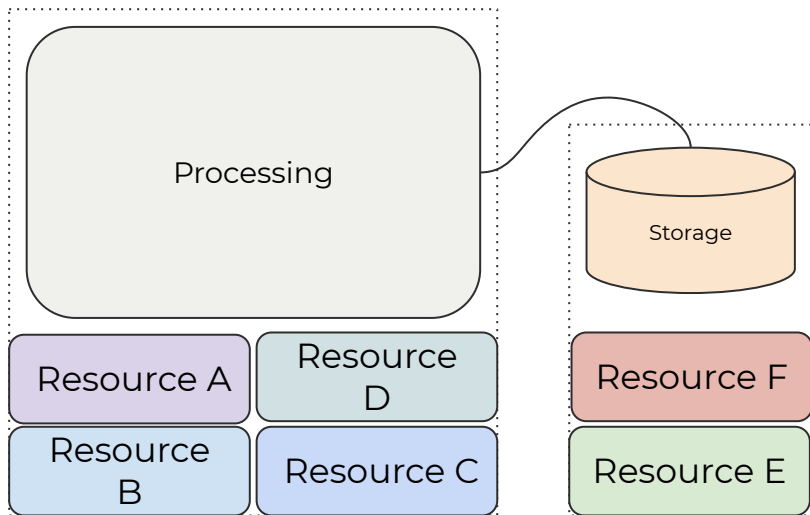
The Monolith

In the beginning, software was implemented as one chunk of code.

Monoliths in its purest form, does not scale horizontally, only vertically.



The nature of a monolith



Deployment

A classic monolith has all code deployed at the same time, in one chunk.

Underlying resources

With VM technology you CAN add more hardware to one VM, but at some point it will stop adding value.

An accident waiting to happen

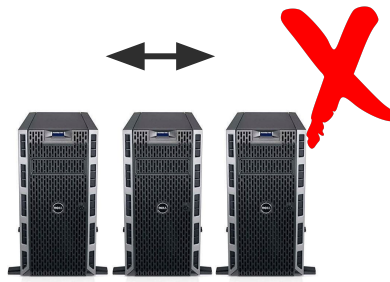
As a monolith only scales vertically, there is a limit to how much you can scale it. At some point, the battle is lost. Horizontal scaling is extremely complex.

The monolith



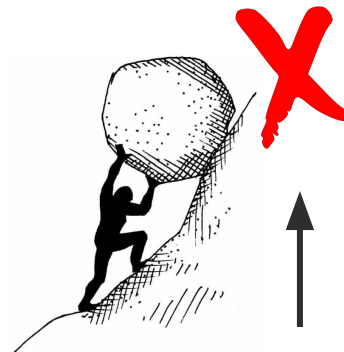
Vertical scaling

Possible to the extent
of the underlying
server



Horizontal scaling

Virtually impossible
with monolith -
technically
challenging

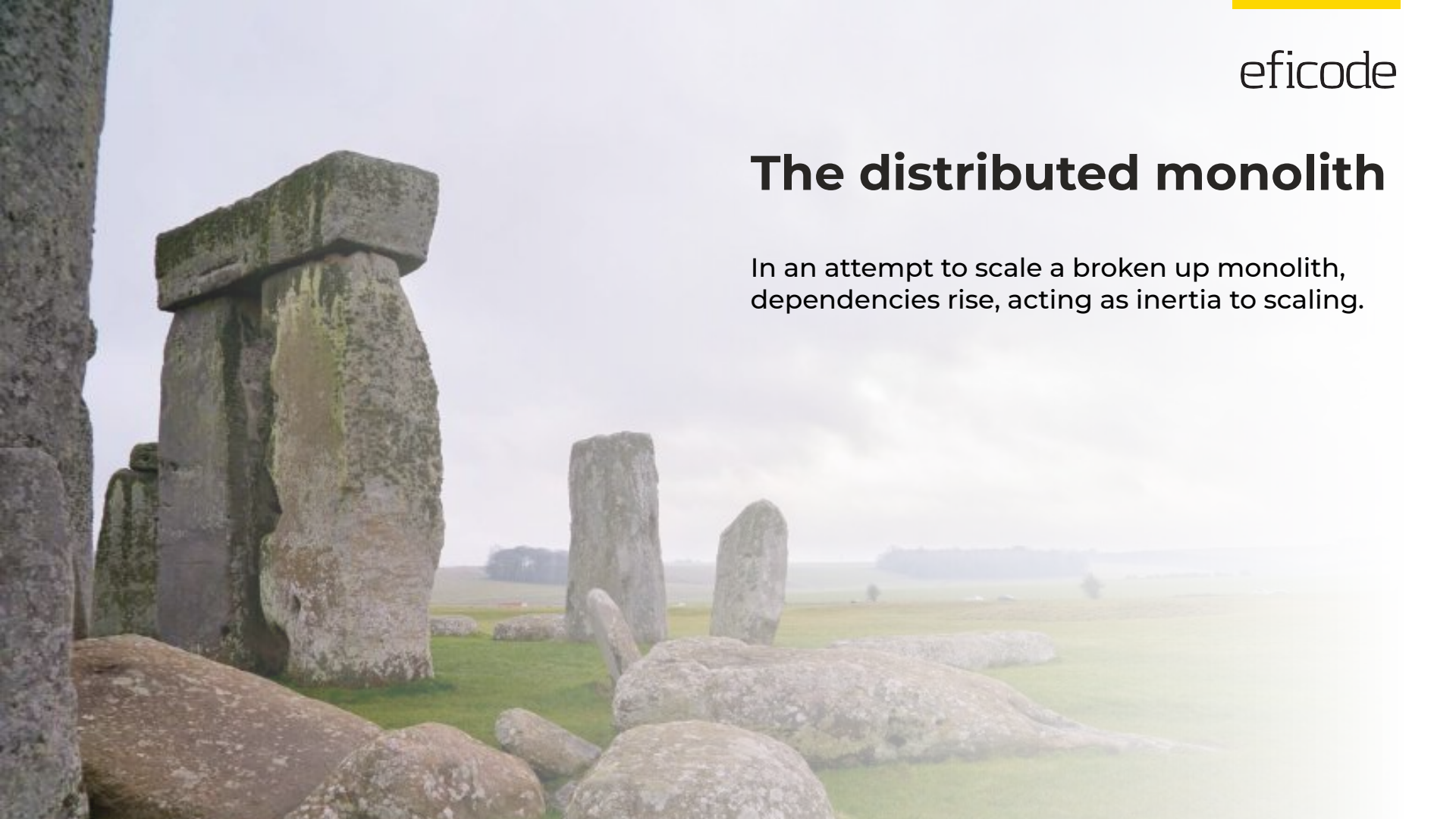


Inertia building

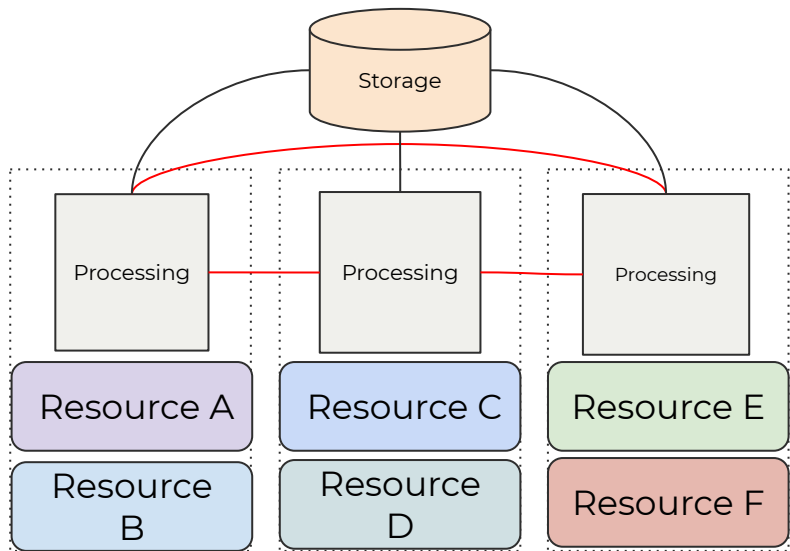
As the monolith grows,
Inertia grows as well,
making scaling
harder

The distributed monolith

In an attempt to scale a broken up monolith, dependencies rise, acting as inertia to scaling.



The nature of a distributed monolith



— Dependencies

Splitting the monolith

A distributed monolith is simply a monolith that has been split into multiple different parts.

Underlying resources

It can be deployed to multiple VM's as it's no longer one chunk of code.

Still a monolith

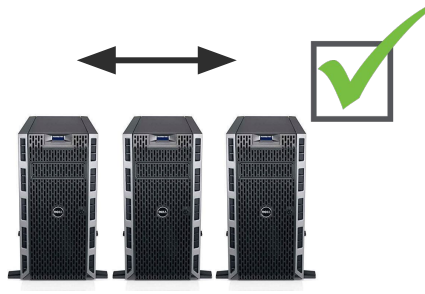
It's still a monolith as every part is hard coupled to each other, and expect the other parts to always be up. So an upgrade to one part, will mean downtime for those depending on it.

The distributed monolith



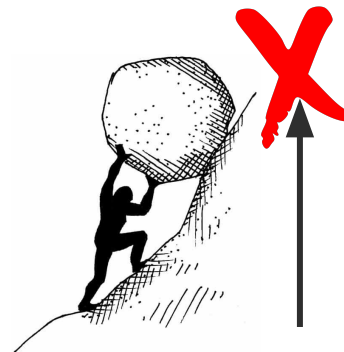
Vertical scaling

Possible to the extent
of the underlying
server



Horizontal scaling

Easier to scale than a
classic monolith.



Inertia building

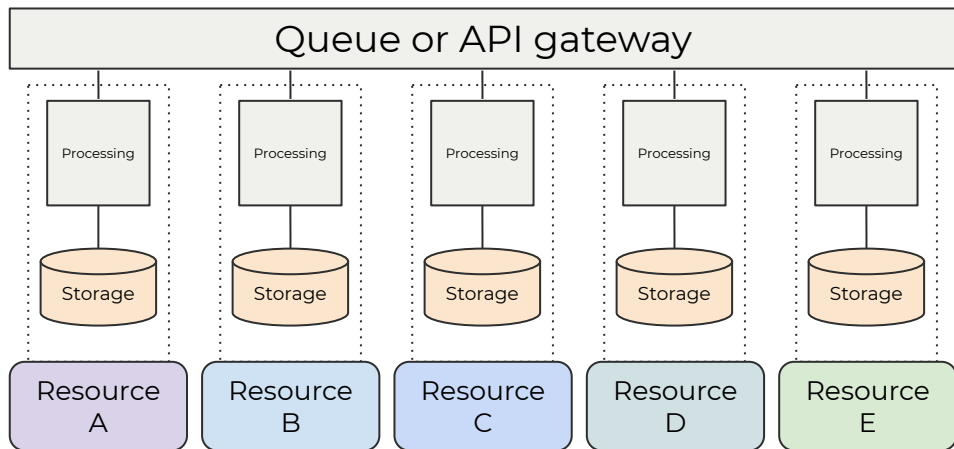
Inertia builds
up rapidly, as
dependencies
between services
keep growing

Decoupled services

Decoupled services does not depend on one another directly.

They are independently deployable, scalable and don't build up inertia.

The nature of decoupled services



Removed dependencies

Every service is only depending on a queue or API gateway solving the dependency problem of monoliths.

Clean interfaces is a must

Hard contracts between services and the queue allows them to scale and work independently of each other.

Scaling

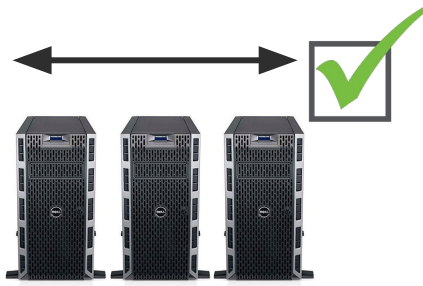
These services can be scaled horizontally and vertically without any inertia building up

Decoupled services



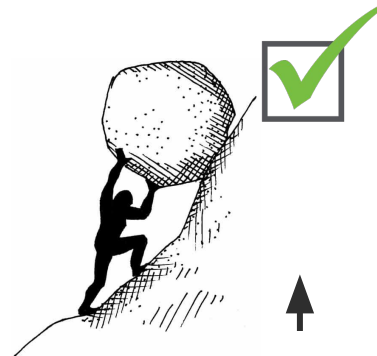
Vertical scaling

Possible to the extent
of the underlying
server



Horizontal scaling

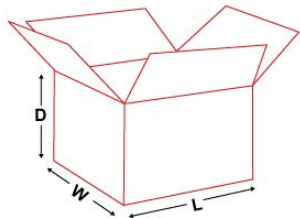
Easier to scale than
any monolith



Inertia building

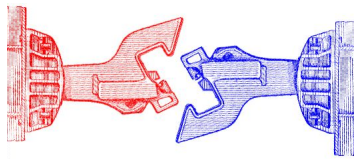
Inertia is not present
in the code, as it is
moved to the
decoupling layer.

Three rules for decoupling



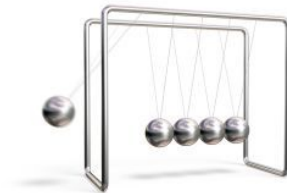
Services size

The size of a service should never be bigger than the team implementing it



Decouple it

You need to decouple your services with something stable, like a queue or API gateway



React on

Services should react on events only

```
size + NGROUPS_PER_BLOCK - 1) / NGROUPS_PER_BLOCK;  
/* always allocate at least one indirect block pointer */  
: 1;  
c(sizeof(*group_info) + nblocks*sizeof(gid_t *), GFP_USER);  
  
= gidsetsize;  
= nblocks;  
info->usage, 1);
```

ACCESS GRANTED

```
t_undo_partial_alloc;  
>blocks[i] = b;
```

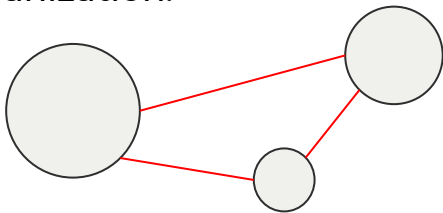
Let's hack Conway

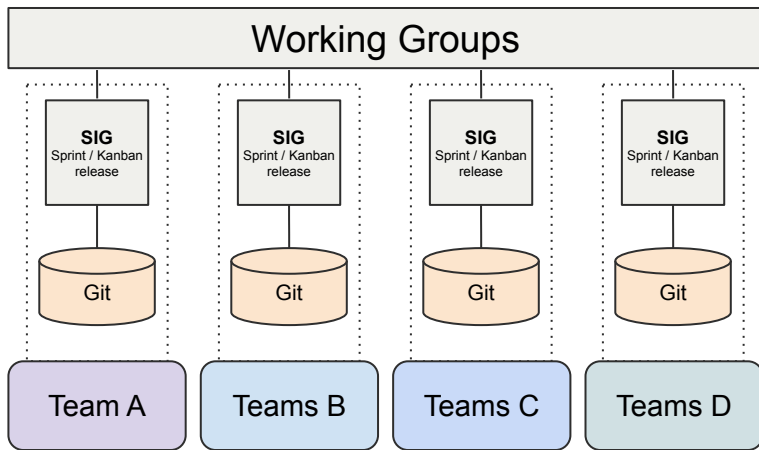
- doing an inverse Conway

Monolithic architectures

Monoliths, being classic or distributed, will introduce dependencies between the teams that need to implement it. It's inevitable.

These dependencies are the core problem when scaling an organization.



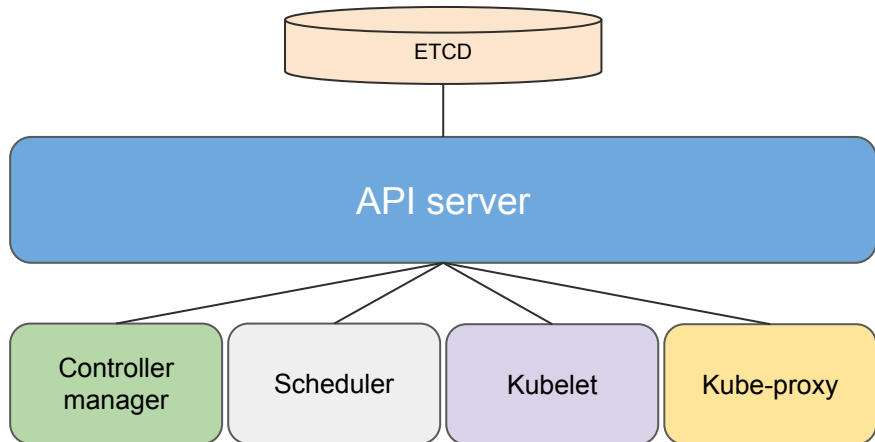


Decoupled architectures

Decoupled architectures don't build up dependencies between teams, as these are dealt with in planning groups instead of in the team that implements it.

In open-source they are called Working Groups and SIG (Special Interest Group)

https://en.wikipedia.org/wiki/Special_Interest_Group



An example

Kubernetes is the second largest project on GitHub. Every service is decoupled from the others via the API server.

The other services does not need to be running, and each service does not need to know about the others.

Evolution of growth

eficode



One team

In the beginning, one team can start a project and deliver value

Eg. Kubernetes

Zero scale

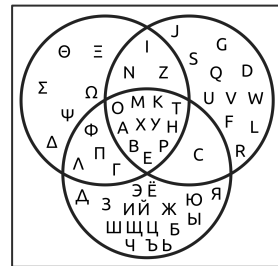


Component

When the team is too big, it can be split up into components

Eg. API-Server, Kubelet, Scheduler

Small scale

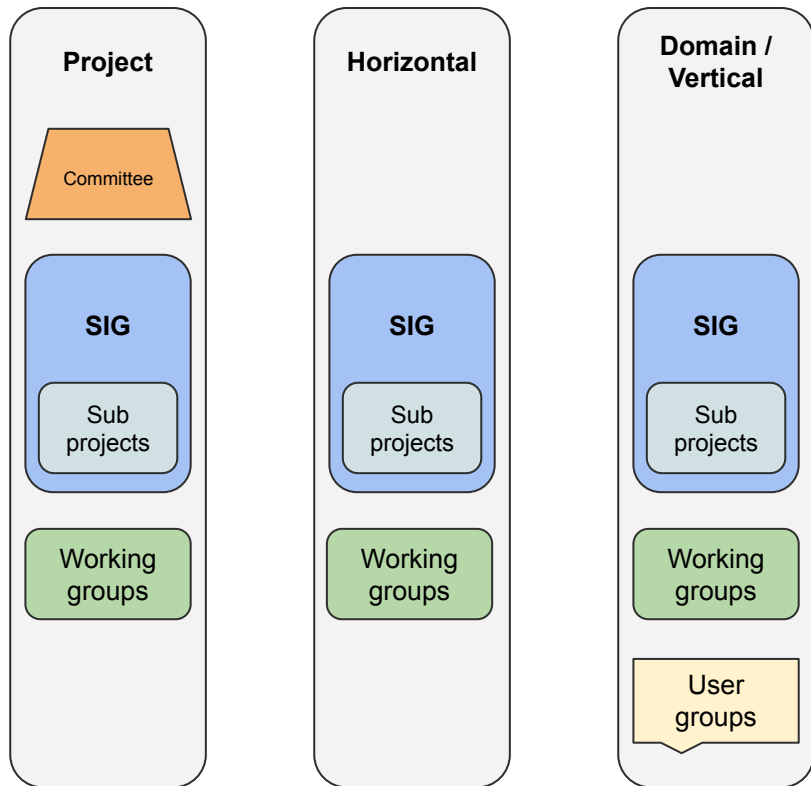


Topics

When the components become too crowded, we break the system into topics

Eg. Auth, Docs, Network

Large scale



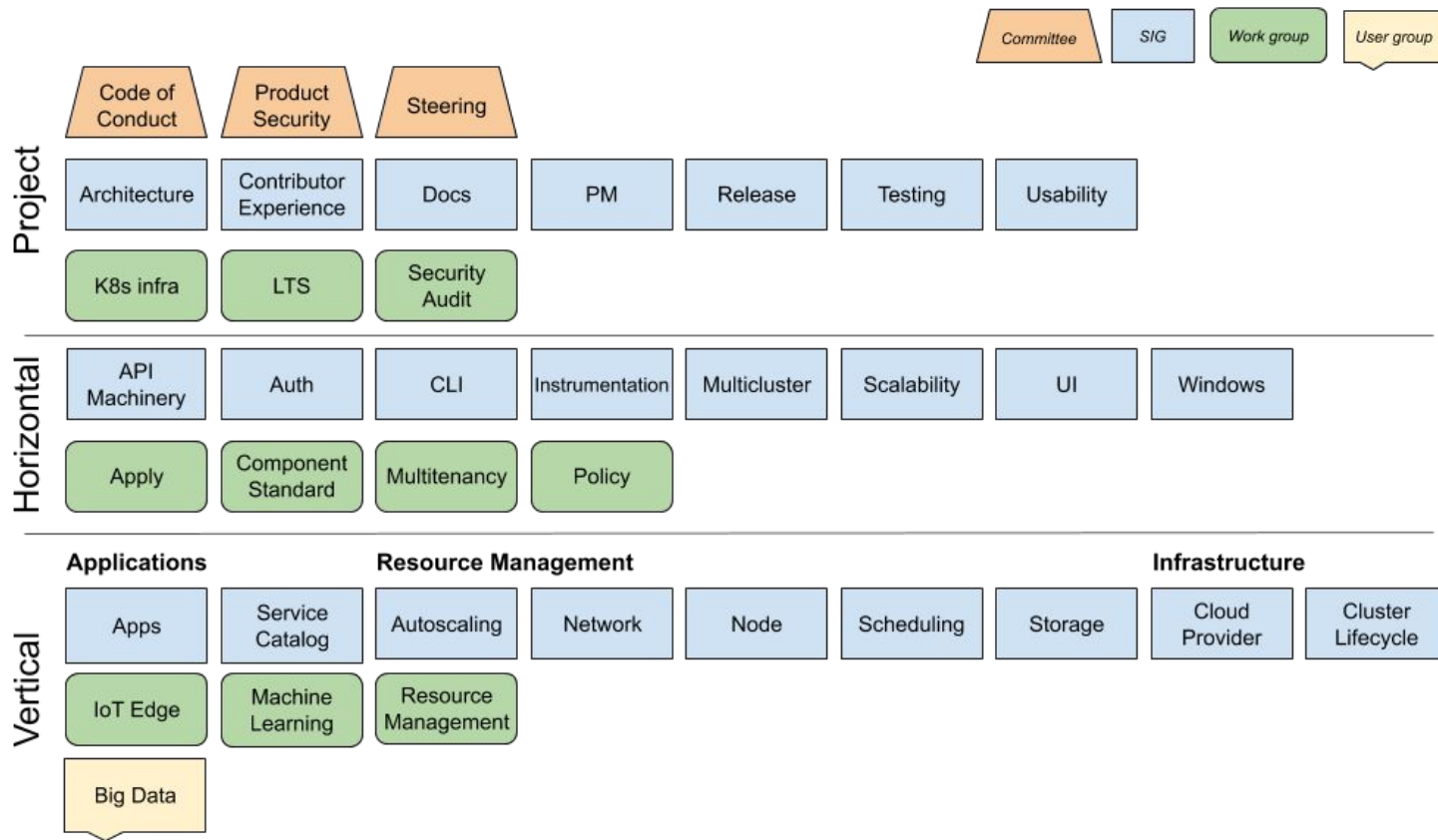
Flat hierarchy

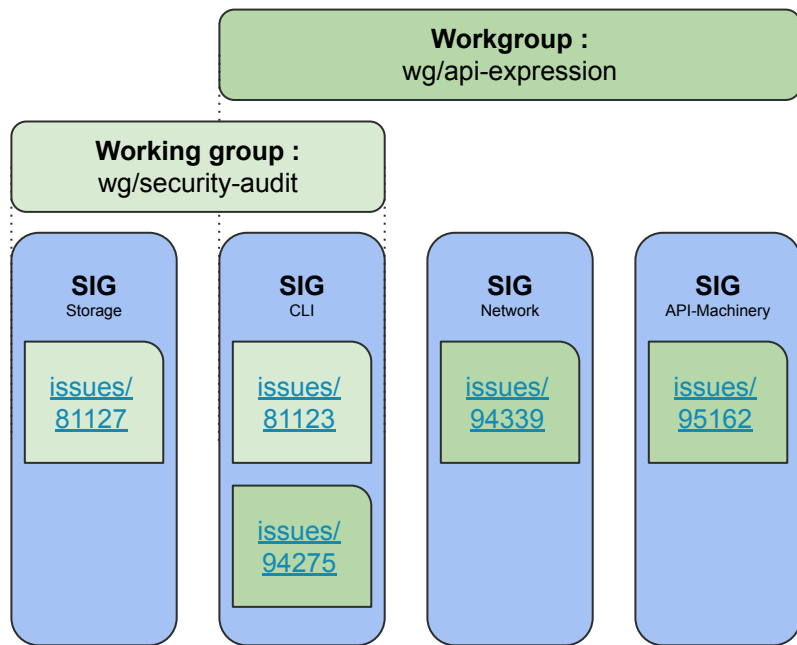
Committees are named sets of people that are chartered to take on sensitive topics.

Special Interest Groups (SIGs) are persistent open groups that focus on a part of the project.

Subprojects are smaller groups that can work independently.

Working Groups are temporary groups that are formed to address issues that cross SIG boundaries. Working groups do not own any code or other long term artifacts.





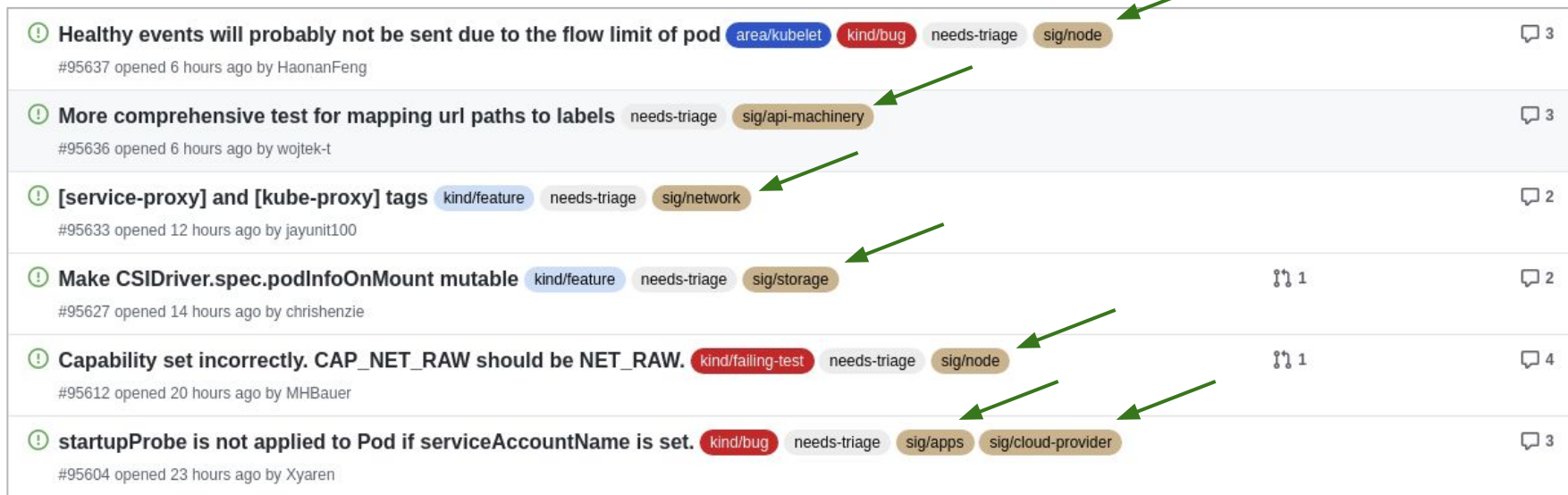
Mapping tasks

We can now map issues and tasks all the way from workgroup to an issue in a SIG group.

Some issues are bugs that gets assigned directly to a SIG, others from a workgroup.

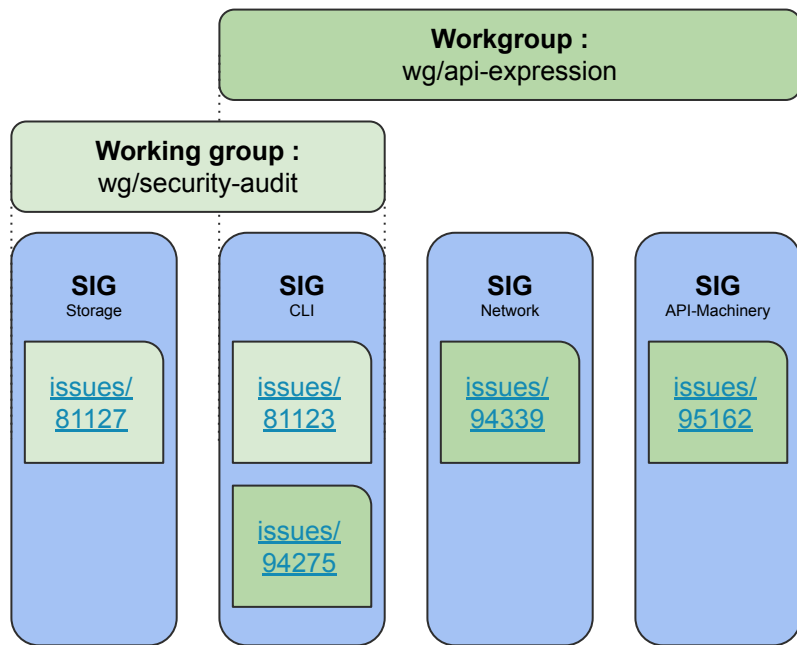
Only SIG owns code !

SIG/WG Issues



A screenshot of a list of GitHub issues. Each issue entry includes a title, a status icon (green exclamation mark), a list of labels, and a comment count. Green arrows point from the right side of the image to specific labels in the issues: 'sig/node' in the first issue, 'sig/api-machinery' in the second, 'sig/network' in the third, 'sig/storage' in the fourth, 'sig/node' in the fifth, and 'sig/apps' and 'sig/cloud-provider' in the sixth.

! Healthy events will probably not be sent due to the flow limit of pod	area/kubelet	kind/bug	needs-triage	sig/node	3
#95637 opened 6 hours ago by HaonanFeng					
! More comprehensive test for mapping url paths to labels	needs-triage	sig/api-machinery			3
#95636 opened 6 hours ago by wojtek-t					
! [service-proxy] and [kube-proxy] tags	kind/feature	needs-triage	sig/network		2
#95633 opened 12 hours ago by jayunit100					
! Make CSIDriver.spec.podInfoOnMount mutable	kind/feature	needs-triage	sig/storage		2
#95627 opened 14 hours ago by chrishenzie					
! Capability set incorrectly. CAP_NET_RAW should be NET_RAW.	kind/failing-test	needs-triage	sig/node		4
#95612 opened 20 hours ago by MHBauer					
! startupProbe is not applied to Pod if serviceAccountName is set.	kind/bug	needs-triage	sig/apps	sig/cloud-provider	3
#95604 opened 23 hours ago by Xyaren					

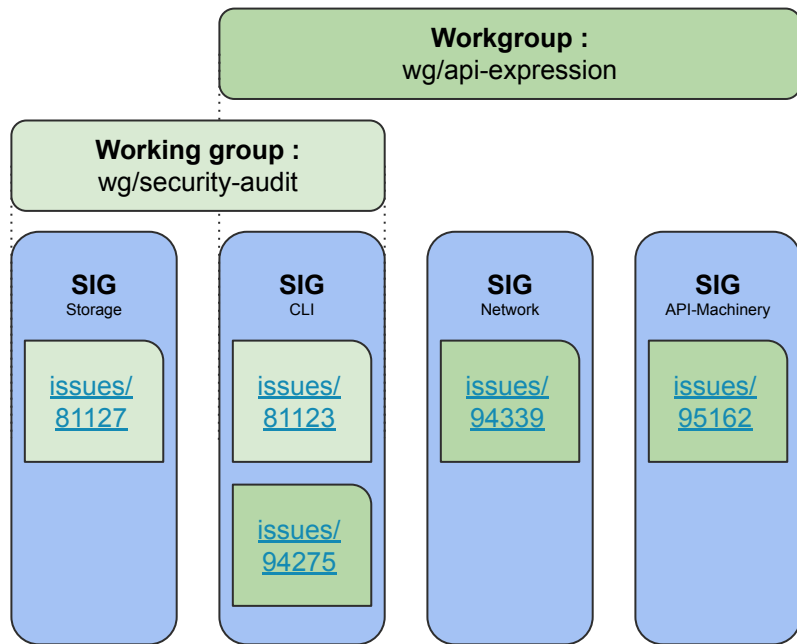


Converting to Jira

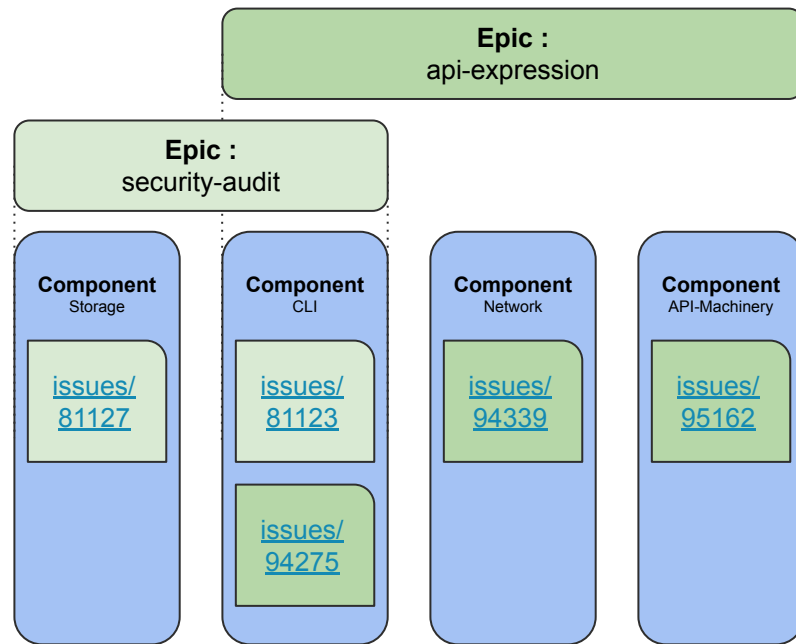
Could we map this from GitHub into Jira?

- Workgroups only exists as labels
- SIG only exist as labels
- Only SIG owns code
- Issues without a WG or SIG label are invalid

GitHub



Jira





**It's all about
distributing decisions.**

*Sigs and the Kubernetes
community - Joe Beda*



... and decoupling them from execution

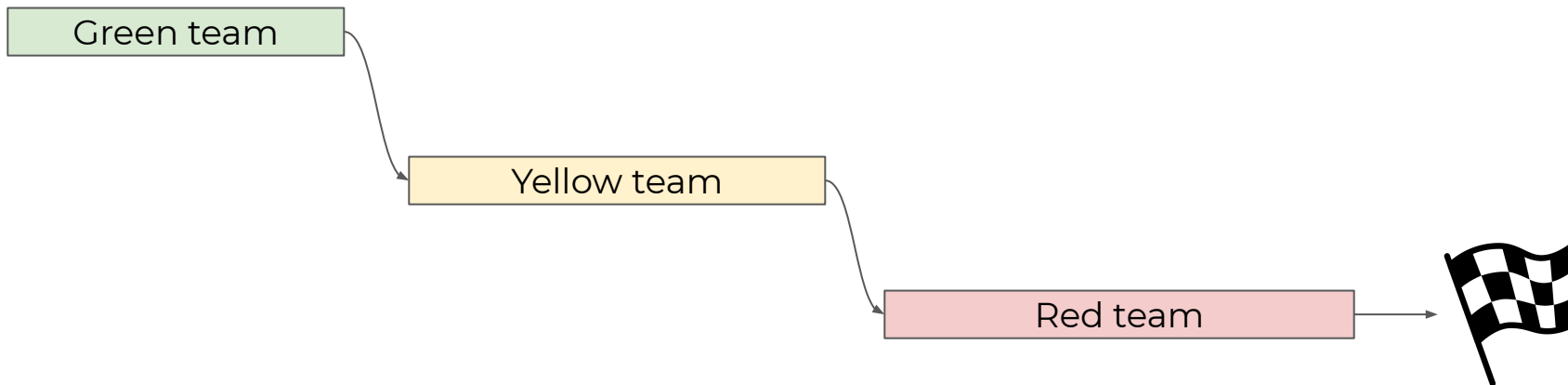
The guy on stage - Henrik Høegh



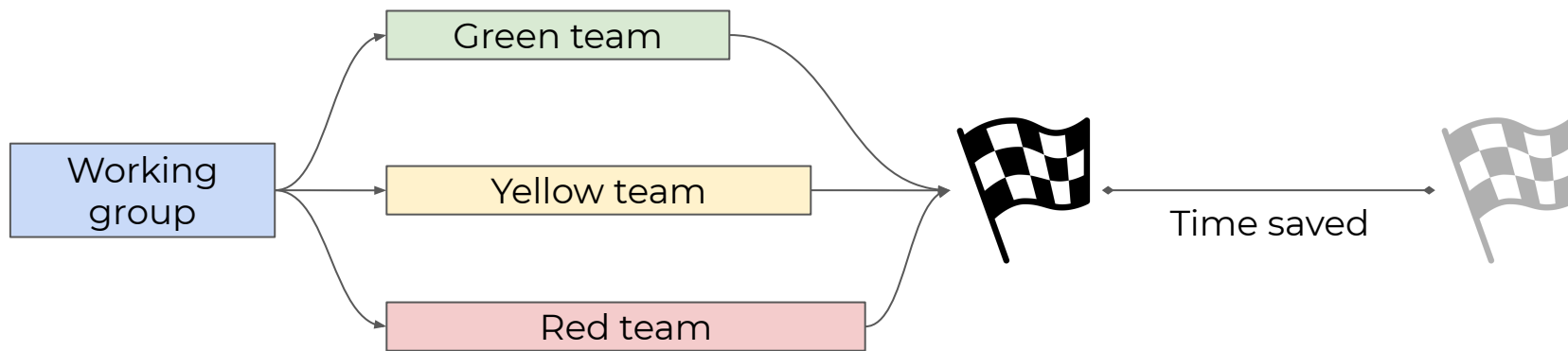
Takeaways

What does this look like?

We don't want hard coupled teams



We want them decoupled so they can deliver independently



It's not about speed

It's about making teams
independently deployable



How team sizes fit

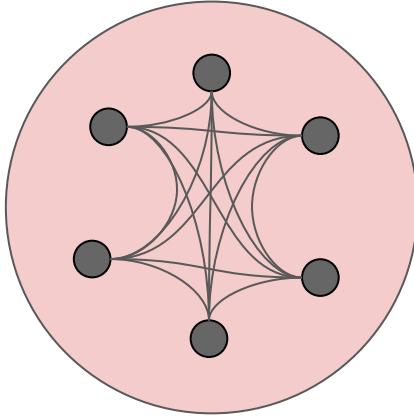


AKA the startup

eficode

Red team

6 people



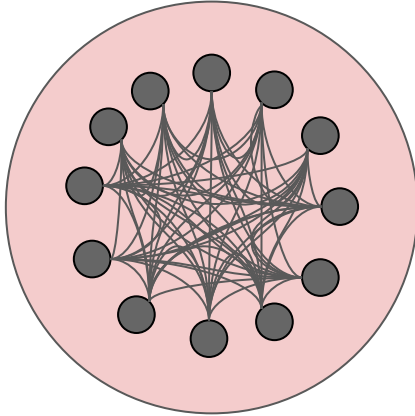
15 connections

AKA the monolith

eficode

Red team

12 people

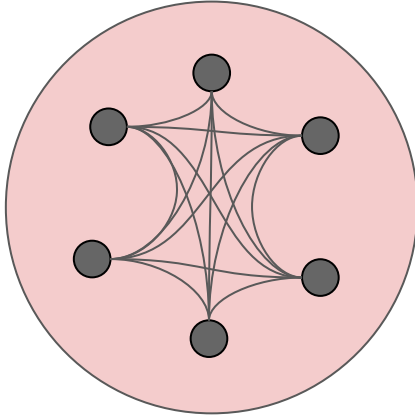


66 connections

AKA the silos

Red team

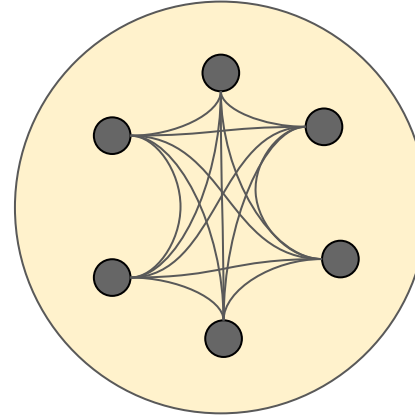
6 people



15 connections

Yellow team

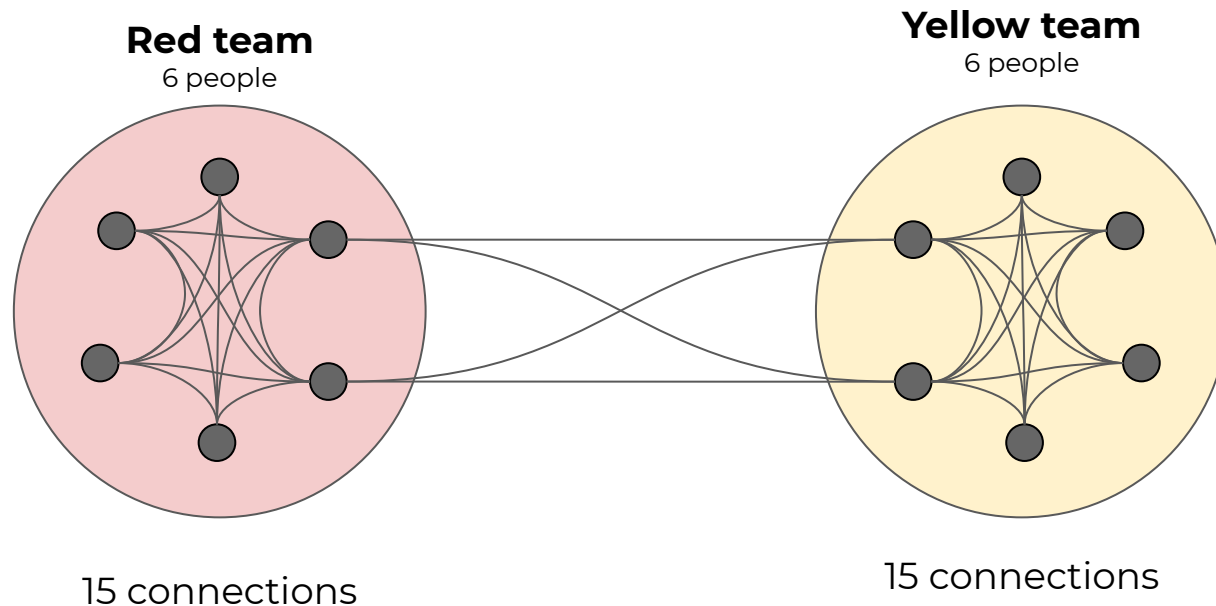
6 people



15 connections

30 connections, but **no** communication

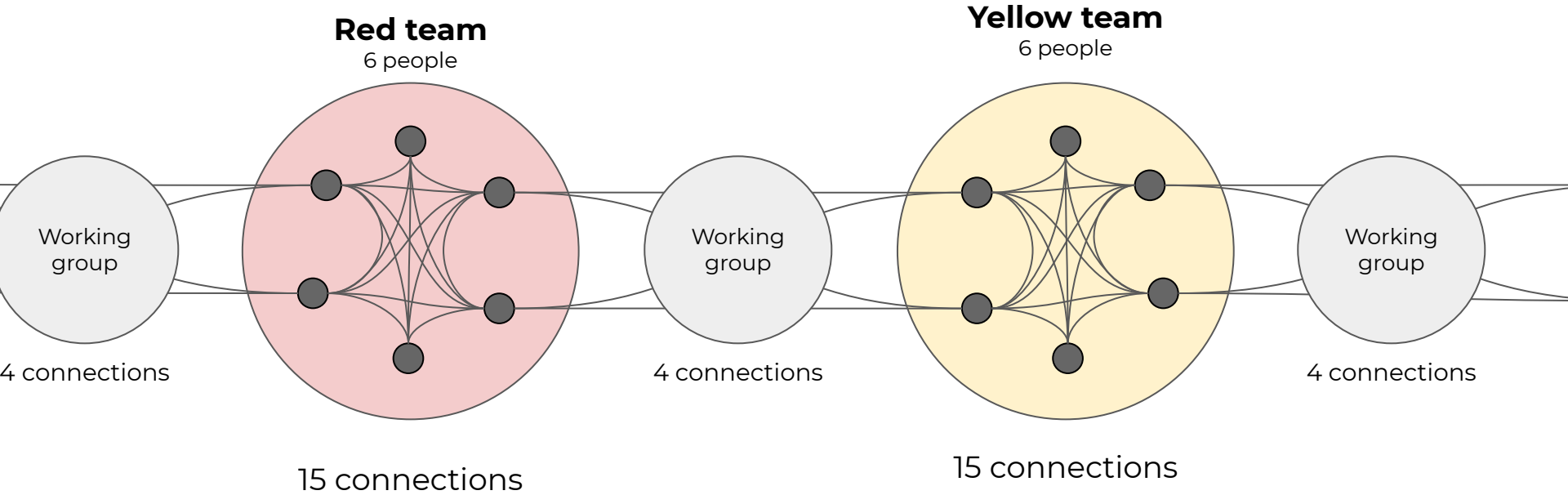
AKA the distributed monolith



34 connections, **with** communication

AKA the decoupled system

eficode

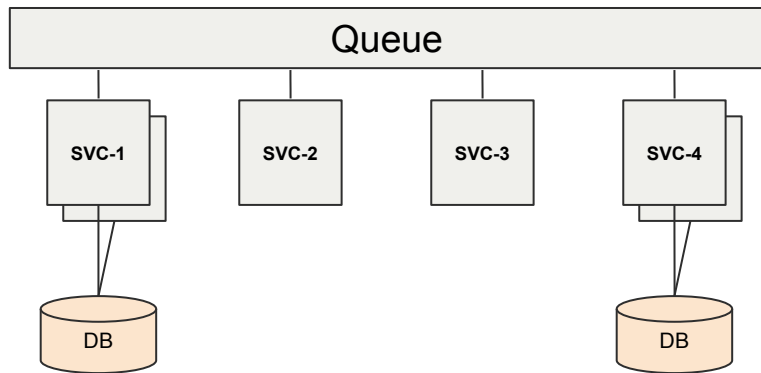


Multiple teams

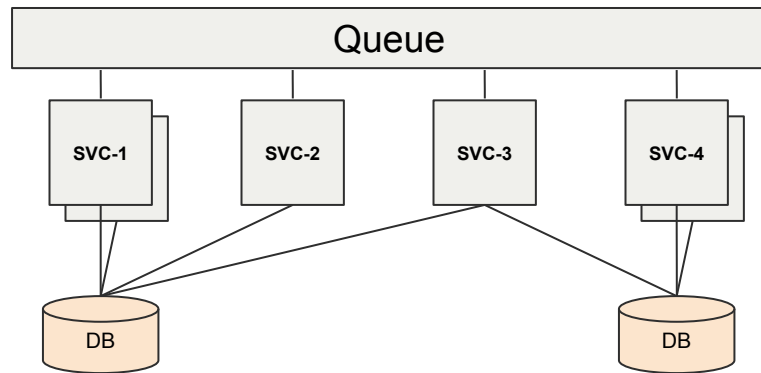
Owning the same code

Basically

We want this

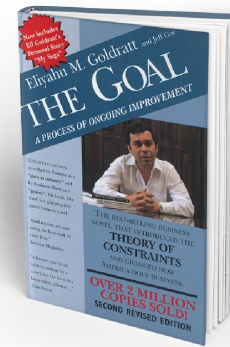


Not this



But

Utilization will drop then ?

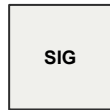


What we do



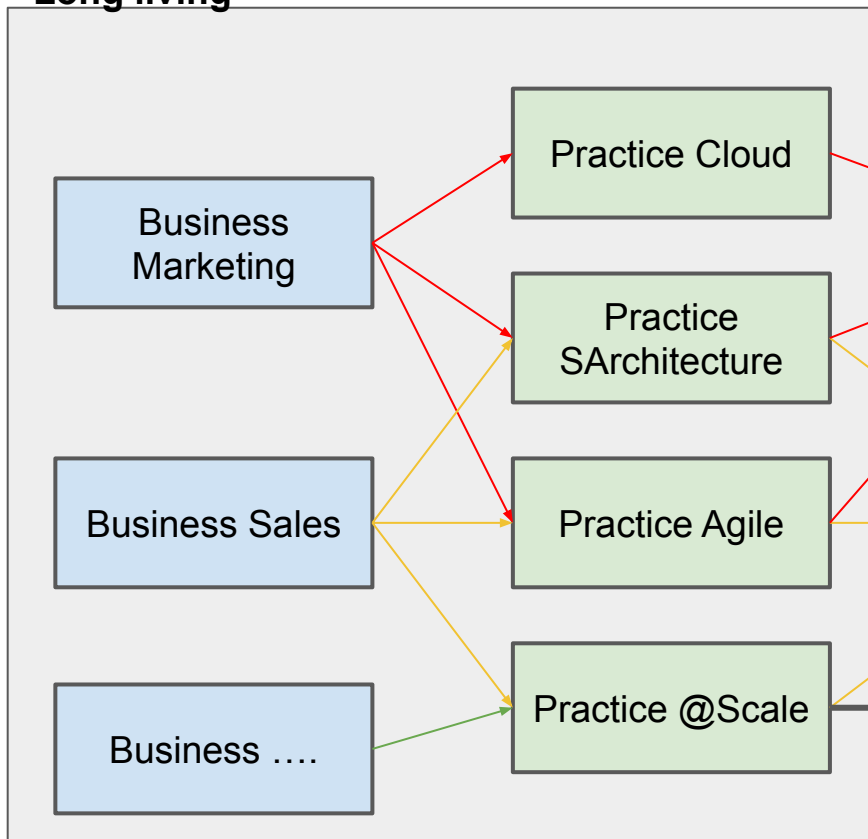
We build communities

And have owners

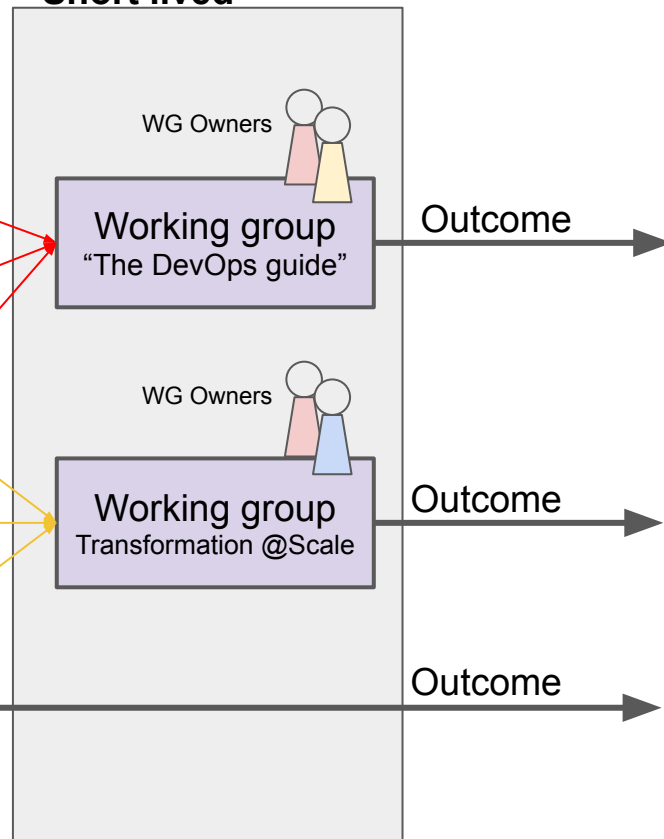


= community domain

Long living



Short lived



Thank you!



Henrik René Høegh

@ henrik.hoegh@eficode.com

 [HenrikHoegh](#)

 +45 25 188 420

Don't forget to
vote for this session
in the **GOTO Guide app**